

Efficient algorithm for the vertex connectivity of trapezoid graphs

ALEKSANDAR ILIĆ [‡]

Faculty of Sciences and Mathematics, Višegradska 33, 18000 Niš, Serbia

e-mail: aleksandari@gmail.com

June 16, 2011

Abstract

The intersection graph of a collection of trapezoids with corner points lying on two parallel lines is called a trapezoid graph. These graphs and their generalizations were applied in various fields, including modeling channel routing problems in VLSI design and identifying the optimal chain of non-overlapping fragments in bioinformatics. Using modified binary indexed tree data structure, we design an algorithm for calculating the vertex connectivity of trapezoid graph G with time complexity $O(n \log n)$, where n is the number of trapezoids. Furthermore, we establish sufficient and necessary condition for a trapezoid graph G to be bipartite and characterize trees that can be represented as trapezoid graphs.

Keywords: trapezoid graphs; vertex connectivity; algorithms; binary indexed tree.

AMS Classifications: 05C85, 68R10, 05C40.

1 Introduction

A trapezoid diagram consists of two horizontal lines and a set of trapezoids with corner points lying on these two lines. A graph $G = (V, E)$ is a trapezoid graph when a trapezoid diagram exists with trapezoid set T , such that each vertex $i \in V$ corresponds to a trapezoid $T[i]$ and an edge exists $(i, j) \in E$ if and only if trapezoids $T[i]$ and $T[j]$ intersect within the trapezoid diagram. A trapezoid $T[i]$ between these lines has four corner points $a[i]$, $b[i]$, $c[i]$ and $d[i]$ – which represent the upper left, upper right, lower left and lower right corner points of trapezoid i , respectively. No two trapezoids share a common endpoint (see Figure 1).

Trapezoid graphs were first investigated by Corneil and Kamula [6]. These graphs and their generalizations were applied in various fields, including modeling channel routing problems in VLSI design [8] and identifying the optimal chain of non-overlapping fragments in bioinformatics [1]. Given some labeled terminals on the upper and lower side of a two-sided channel, terminals with the same label will be connected in a common net. Each net can be modeled by a trapezoid that connects rightmost and leftmost terminals of that net on two horizontal lines. In the channel routing problem we want to connect all terminals of each net so that no two nets intersect. One can show [8] that two nets can be routed without intersection in the same layer if and only if their corresponding trapezoids do not intersect. Therefore, the number of colors needed to color the trapezoid graph is the number of layers needed to route the nets without intersection.

Let n and m denote the number of vertices and edges of a trapezoid graph G . Ma and Spinrad [22] showed that trapezoid graphs can be recognized in $O(n^2)$ time, while Mertzios and Corneil [23] designed structural trapezoid recognition algorithm based on the vertex splitting method in

$O(n(m + n))$ time, which is easier for implementation. Trapezoid graphs are perfect, subclass of cocomparability graphs and properly contain both interval graphs and permutation graphs. If $a[i] = b[i]$ and $c[i] = d[i]$ then the corresponding trapezoid $T[i]$ reduces to a straight line, and the trapezoid graph reduces to a permutation graph if the condition holds for all $1 \leq i \leq n$. Similarly, the trapezoid graph reduces to the interval graph if $a[i] = c[i]$ and $b[i] = d[i]$ for all i .

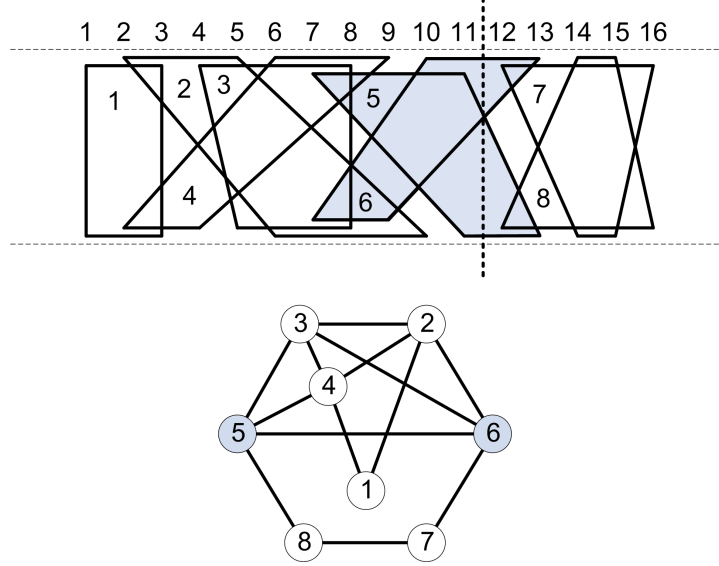


Figure 1: A trapezoid graph and its trapezoid representation (marked vertices represent 2-cut).

Many common graph problems, such as minimum connected dominating sets [25], all-pair shortest paths [24], maximum weighted cliques [2], all cut vertices [13], chromatic number and clique cover [10], all hinge vertices [3] in trapezoid graphs, can be solved in polynomial time. For other related problems see [4, 7, 17, 18]. Recently, Lin and Chen [19] presented $O(n^2)$ algorithms for counting the number of vertex covers (VC), minimal VCs, minimum VCs and maximum minimal VCs in a trapezoid graph. Ilić and Ilić [14] improved algorithms for calculating the size and the number of minimum vertex covers (or independent sets), as well as the total number of vertex covers, and reduce the time complexity to $O(n \log n)$. Ghosh and Pal [11] presented an efficient algorithm to find the maximum matching in trapezoid graphs, which turns out to be not correct [14].

Let $G = (V, E)$ be a simple undirected graph with $|V| = n$. A vertex cut or separating set of a connected graph G is a set of vertices whose removal disconnects G . The connectivity or vertex connectivity $\kappa(G)$ (where G is not complete) is the size of a smallest vertex cut. A graph is called k -connected or k -vertex-connected if its vertex connectivity is greater than or equal to k . A complete graph with n vertices, denoted by K_n , has no vertex cuts, but by convention $\kappa(K_n) = n - 1$. A connected graph is said to be separable if its vertex connectivity is one. In that case, a vertex which disconnects the graph is called a cut-vertex or an articulation point.

The edge cut of G is a group of edges whose total removal disconnects the graph. The edge-connectivity $\lambda(G)$ is the size of a smallest edge cut. In the simple case in which cutting a single edge would disconnect the graph, that edge is called a bridge. Let $\delta(G)$ be minimum vertex degree of G , then

$$\kappa(G) \leq \lambda(G) \leq \delta(G).$$

Since the strength of the network G is proportional to $\kappa(G)$, graph connectivity is one of the most fundamental problem in graph theory. Even and Tarjan [9] have obtained $O(\kappa \cdot mn\sqrt{n})$ time sequential algorithm for finding vertex connectivity of a general graph. The authors in

[21] obtained parallel algorithm for testing k -vertex connectivity in interval graphs. Ghosh and Pal in [12] presented rather complicated algorithm with a lot of different cases to solve the vertex connectivity problem, which takes $O(n^2)$ time and $O(n)$ space for a trapezoid graph. In this paper, we designed an algorithm with time complexity $O(n \log n)$ for calculating the vertex connectivity of a trapezoid graph.

The rest of the paper is organized as follows. In Section 2 we introduce the modified binary indexed tree data structure. In Section 3 we design $O(n \log n)$ time algorithm for calculating the vertex connectivity of trapezoid graphs, improving the algorithm from [12]. In Section 4 we establish sufficient and necessary condition for a trapezoid graph G to be bipartite and characterize trees that can be represented as trapezoid graphs. We close the paper in Section 5 by proposing topics for the further research.

2 Modified binary indexed data structure

The binary indexed tree (BIT) is an efficient data structure for maintaining the cumulative frequencies. We will modify this standard structure to work with minimal/maximal partial summations.

Let A be an array of n elements. The modified binary indexed tree (MBIT) supports the following basic operations:

- (i) for given value x and index i , add x to the element $A[i]$, $1 \leq i \leq n$;
- (ii) for given interval $[1, i]$, find the sum of values $A[1], A[2], \dots, A[i]$, $1 \leq i \leq n$.
- (iii) for given interval $[1, i]$, find the minimum value among partial sums $A[1], A[1] + A[2], A[1] + A[2] + A[3], \dots, A[1] + A[2] + \dots + A[i]$, $1 \leq i \leq n$.

Naive implementation of these operations have complexities $O(1)$, $O(n)$ and $O(n)$, respectively. We can achieve better complexity, if we speed up the second and third operation which will also affect the first operation.

The main idea of the modified binary indexed tree structure is that sum of elements from the segment $[1, i]$ can be represented as sum of appropriate set of subsegments. The MBIT structure is based on decomposition of the cumulative sums into segments and the operations to access this data structure are based on the binary representation of the index. This way the time complexity for all operations will be the same $O(\log n)$.

The structure is a complete binary tree with the root node 1. Its leafs correspond to the elements from the array A , starting from left to right in the last level. Therefore, the elements of the array A are stored at the positions starting from 2^p to $2^p + n - 1$, where p is the depth of the binary tree (defined as the smallest integer such that $2^p \geq n$). The internal nodes store the cumulative values of the leafs in the subtrees rooted at these nodes. This implies that the value of the internal node i is just the cumulative value of its two children. The parent of the node i is $\lfloor \frac{i}{2} \rfloor$, while the left and the right child of the node i are $left[i] = 2i$ and $right[i] = 2i + 1$, respectively. By definition, it follows that the number of nodes in MBIT is at most $2n$, while the depth is $\lceil \log_2 n \rceil$.

In addition to the values of the array A , for each node we will keep two information: $sum[x]$ will be the sum of the $A[i]$ values of all nodes in x 's subtree, and $min_sum[x]$ will be the minimum possible cumulative sum of $A[i]$'s in the subtree rooted at x (starting at the leftmost node in the subtree). We will demonstrate how to compute these fields using only information at each node and its children. The sum of the $A[i]$'s of the subtree rooted at node x will simply be

$$sum[x] = sum[2x] + sum[2x + 1].$$

The minimum cumulative sum can either be in the left subtree or in the right subtree. If it is in the left subtree, it is simply $\text{min_sum}[\text{left}[x]]$, while if it is in the right subtree, we have to add the cumulative sum up till the right subtree to the minimum value of the right subtree $\text{sum}[\text{left}[x]] + \text{min_sum}[\text{right}[x]]$. Finally, we get

$$\text{min_sum}[x] = \min(\text{min_sum}[2x], \text{sum}[2x] + \text{min_sum}[2x + 1]).$$

For the update procedure, we just need to traverse the vertices from the leaf to the root and update the values in the parent vertices based on the above formulas. For the query procedure, we traverse the binary tree in a top-down manner starting from the root vertex 1. The important thing is to maintain the partial sums of the array A (starting from $A[1]$). If the leaf that stores $A[\text{index}]$ belongs to the left child – we just go left and do nothing; otherwise we update the partial sum of the left subtree and the index, and go right.

For detailed implementation see Algorithms 1 and 2. The structure is space-efficient in the sense that it needs the same amount of storage as just a simple array of n elements. Furthermore we can use fast bitwise operations (xor, and, shift left) for more efficient implementation.

Theorem 1 *Calculating the sum of the elements from $A[1]$ to $A[i]$, calculating the minimum value among partial sums $A[1], A[1] + A[2], \dots, A[1] + A[2] + \dots + A[i]$, and updating the element $A[i]$ in the modified binary indexed tree is performed in $O(\log n)$ time, $1 \leq i \leq n$.*

Algorithm 1: Updating the modified binary indexed tree.

Input: The value value , the element index index and the parameters n and

$$p = \min\{s : 2^s \geq n\}.$$

```

1  $i = \text{index} + 2^p - 1;$ 
2  $\text{sum}[i] = \text{value};$ 
3  $\text{min\_sum}[i] = \text{value};$ 
4 while  $i > 1$  do
5   if  $i \bmod 2 = 1$  then
6      $i = i - 1;$ 
7   end
8    $\text{sum}[i/2] = \text{sum}[i] + \text{sum}[i + 1];$ 
9    $\text{min\_sum}[i/2] = \min(\text{min\_sum}[i], \text{sum}[i] + \text{min\_sum}[i + 1]);$ 
10   $i = i/2;$ 
11 end
```

This approach is very similar to the problem regarding calculating the point of maximum overlap among intervals (see [5] Problem 14-1), that can be solved using red-black trees.

Example 2 *Let $n = 14$ and $p = 4$. The elements of the array A will be stored on positions $\text{sum}[16]$ to $\text{sum}[29]$ and $\text{min_sum}[16]$ to $\text{min_sum}[29]$. The node 5 will contain the following information*

$$\text{sum}[5] = \text{sum}[20] + \text{sum}[21] + \text{sum}[22] + \text{sum}[23] = A[5] + A[6] + A[7] + A[8]$$

$$\text{min_sum}[5] = \min(A[5], A[5] + A[6], A[5] + A[6] + A[7], A[5] + A[6] + A[7] + A[8]).$$

If we change the value $A[6]$, we will start from the corresponding index $i = 21$ and change the following nodes of the modified binary index tree: 21, 10, 5, 2 and 1. If we want to calculate the minimum among partial sums with elements $A[1], A[2], \dots, A[13]$, we calculate the following minimum

$$\min(\text{min_sum}[2], \text{sum}[2] + \text{min_sum}[6], \text{sum}[2] + \text{sum}[6] + \text{min_sum}[28]).$$

Algorithm 2: Calculating the minimal cumulative partial sum.

Input: The index $index$.

Output: The minimum among partial sums with elements $A[1], A[2], \dots, A[index]$.

```
1  $min = \infty$ ;
2  $partial\_sum = 0$ ;
3  $i = 1$ ;
4  $pow = 2^{p-1}$ ;
5 while  $index > 0$  do
6   if  $index \geq pow$  then
7     if  $min > partial\_sum + min\_sum[2 \cdot i]$  then
8        $min = partial\_sum + min\_sum[2 \cdot i]$ ;
9     end
10     $partial\_sum = partial\_sum + sum[2 \cdot i]$ ;
11     $i = 2 \cdot i + 1$ ;
12     $index = index - pow$ ;
13  end
14  else
15     $i = 2 \cdot i$ ;
16  end
17   $pow = pow/2$ ;
18 end
19 return  $min$ ;
```

3 The algorithm for the vertex connectivity

Let $T = \{1, 2, \dots, n\}$ denote the set of trapezoids in the trapezoid graph $G = (V, E)$. For simplicity, the trapezoid in T that corresponds to vertex i in V is called trapezoid $T[i]$. Without loss of generality, the points on each horizontal line of the trapezoid diagram are labeled with distinct integers between 1 and $2n$.

Trapezoid i lies entirely to the left of trapezoid j , denoted by $i \ll j$, if $b[i] < a[j]$ and $d[i] < c[j]$. It follows that \ll is a partial order over the trapezoid set T and (T, \ll) is a strictly partially ordered set.

Lemma 3 [12] *Two vertices $T[i]$ and $T[j]$ of a trapezoid graph are not adjacent iff either (i) $b[i] < a[j]$ and $d[i] < c[j]$ or (ii) $b[j] < a[i]$ and $d[j] < c[i]$.*

Define a cut line as line p that passes through the intervals $(x, x + 1)$ and $(y, y + 1)$ on the upper and the bottom horizontal line, respectively, and does not contain the integer points $x, x + 1, y, y + 1$. For a such cut, let $N(x, y)$ be the number of trapezoids that have common points with the line p . Define $N(x, y) = \infty$ if there are no trapezoids completely left and no trapezoids completely right of the line p .

Lemma 4 *Let $G \neq K_n$ be a trapezoid graph. Then*

$$\kappa(G) = \min_{1 \leq x \leq 2n, 1 \leq y \leq 2n} N(x, y).$$

Proof. Let S be a vertex cut of the graph G with the minimum cardinality $\kappa(G)$. The removal of S disconnects G , and consider the component C that contains a trapezoid with the smallest upper left corner $a[i]$. Let x be the maximum value among upper right corners in the component C , $x = \max_{i \in C} b[i]$, and let y be the maximum value among lower right corners in

the component C , $y = \max_{i \in C} d[i]$. Since the right border of trapezoids from C form concave broken line – the line p that passes through intervals $(x, x + 1)$ and $(y, y + 1)$ is one cut of a trapezoid graph. It follows that $\kappa(G) \geq N(x, y)$. The other inequality follows immediately, and this completes the proof. \square

Note that in the above theorem the points $x = 1$ and $x = 2n$, as well as $y = 1$ and $y = 2n$, can be excluded from the consideration.

Therefore, one can traverse all values x and y and compute the number of trapezoids that have non-empty intersection with the line p determined by the intervals $(x, x + 1)$ and $(y, y + 1)$. The important thing is to ensure that there are trapezoids lying entirely to the left and to the right of the line p . This can be easily done in $O(n^2)$.

We will first precompute the leftmost and the rightmost trapezoids for each interval $(x, x + 1)$ from the upper line. Let *leftmost* be the index of a trapezoid with $b[\textit{leftmost}] \leq x$ and minimal lower right corner d . Similarly let *rightmost* be the index of a trapezoid with $a[\textit{rightmost}] \geq x + 1$ and maximal lower left corner c . If there are no such trapezoids, set the values of *leftmost* and *rightmost* to -1 . We need additional arrays *index_up* and *index_bottom*, such that *index_up*[j] contains the index of the trapezoid with the left or right coordinate equal to j on the upper line, and similarly for the bottom line. This can be done in linear time $O(n)$ as shown in Algorithm 3 (implementation for the *rightmost* array is similar and, thus, omitted).

Algorithm 3: Calculating the leftmost trapezoids.

Input: The trapezoids T and the array *index_up*.

Output: The array *leftmost*.

```

1 leftmost[1] = -1;
2 for  $j = 2$  to  $2n$  do
3    $i = \textit{index\_up}[j]$ ;
4    $\textit{leftmost}[j] = \textit{leftmost}[j - 1]$ ;
5   if  $b[i] = j$  then
6     if  $(\textit{leftmost}[j] = -1)$  or  $(d[\textit{leftmost}[j]] > d[i])$  then
7        $\textit{leftmost}[j] = i$ ;
8     end
9   end
10 end
11 return leftmost;
```

We will traverse the coordinates on the upper line from $x = 1$ to $x = 2n$, and skip the values with $\textit{leftmost}[x] = -1$ or $\textit{rightmost}[x] = -1$. For each value y between $d[\textit{leftmost}[x]]$ and $c[\textit{rightmost}[x]]$, we need to calculate the number of trapezoids $N(x, y)$ that cut the line p . The trapezoid $T[i]$ cuts the line p if

- it contains the interval $(x, x + 1)$;
- it is left trapezoid with the lower right corner greater than y , i. e. if $b[i] \leq x$ and $d[i] > y$;
- it is right trapezoid with the lower left corner less than or equal to y , i. e. if $a[i] > x$ and $c[i] \leq y$.

Furthermore, we will maintain the binary array *cut* of length n that indicates whether the trapezoid $T[i]$ contains the interval $(x, x + 1)$. In other words, $\textit{cut}[i] = \textit{true}$ if and only if $a[i] \leq x < x + 1 \leq b[i]$. Since no two trapezoids have a common corner, we can update this array in the constant time by traversing from x to $x + 1$. Therefore, for each trapezoid $T[i]$

with $cut[i] = true$ we easily check whether this trapezoid is on the left of x or on the right of x . We can also keep the number of left and right trapezoids in the variables $left$ and $right$ (see Algorithm 5).

In order to calculate the minimum value $N(x, y)$ for the fixed x coordinate, we will traverse y coordinates, and count the number of trapezoids with $cut[i] = false$ that intersect the line p . The idea is to calculate the cumulative sum by adding $+1$ for each coordinate $c[i]$ of right trapezoids and by adding -1 for each coordinate $d[i]$ of left trapezoids. The starting value of the cumulative sum is $left$. The number $N(x, y)$ is equal to the number of trapezoids that contain the interval $(x, x + 1)$ plus the cumulative sum. The pseudo-code of this approach is given in Algorithm 4.

Algorithm 4: Calculating the minimum value $N(x, y)$ for the given coordinate x .

Input: The trapezoids T and the arrays $leftmost$, $rightmost$, $index_bottom$, cut and parameters x , $left$, $right$.

Output: The minimum value $N(x, y)$ for $1 \leq y \leq 2n$.

```

1   $sum = left$ ;
2   $min\_sum = -1$ ;
3  for  $y = 1$  to  $c[rightmost[x]]$  do
4      if  $(y \geq d[leftmost[x]])$  and  $(y \leq c[rightmost[x]])$  then
5          if  $(min\_sum = -1)$  or  $(min\_sum > sum)$  then
6               $min\_sum = sum$ ;
7          end
8      end
9       $i = index\_bottom[y]$ ;
10     if  $(b[i] \leq x)$  and  $(y = d[i])$  and  $(cut[i] = false)$  then
11          $sum = sum - 1$ ;
12     end
13     if  $(a[i] > x)$  and  $(y = c[i])$  and  $(cut[i] = false)$  then
14          $sum = sum + 1$ ;
15     end
16 end
17 if  $min\_sum > -1$  then
18     return  $(n - left - right) + min\_sum$ ;
19 end
20 else
21     return  $-1$ ;
22 end

```

Example 5 The vertex connectivity of the graph G in Figure 1 is two. For $x = 11$, we have the following parameters $left = 5$, $right = 2$, $cut[6] = true$ and $cut[i] = false$ for $1 \leq i \leq 8$ and $i \neq 6$. The execution of Algorithm 4 is presented in Table 1.

From the above table, we conclude that the minimum value of cumulative sums from $y = 3$ to $y = 13$ equals 1 and it is achieved for $y = 10, 11, 13$. This shows that the vertex connectivity of G is less than or equal to $1 + (8 - 5 - 2) = 2$.

For the efficient implementation, we will use the modified binary indexed tree data structure. The array A will correspond to the lower coordinates from 1 to $2n$. For each trapezoid $T[i]$ with $cut[i] = false$, assign $A[c[i]] = 0$ and $A[d[i]] = -1$ if $T[i]$ is a left trapezoid, and assign $A[c[i]] = 1$ and $A[d[i]] = 0$ if $T[i]$ is a right trapezoid. In order to find the minimum value of

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$leftmost$	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$rightmost$	7	7	7	7	7	7	7	7	7	7	7	8	8	-1	-1	-1
A	0	0	-1	-1	0	0	0	-1	0	-1	0	1	-1	1	0	0
sum	5	5	4	3	3	3	3	2	2	1	1	2	1	2	2	2

Table 1: Example of the algorithm execution.

$N(x, y)$ for $1 \leq y \leq 2n$, we can just return $Calculate(rightmost[x])$. The only problem is to ensure that there is at least one left trapezoid for each unit interval $(x, x+1)$. We can solve this by setting $-n^2 - 1$ for the value $A[d[leftmost[x]]]$ and taking this into account when calculating the minimum values. The number $n^2 + 1$ is big enough and all partial sums before the index $leftmost[x]$ will be greater than n (we update the vertex connectivity only if $N(x, y)$ is less than or equal to n). At the end we need to handle the special case – when G is a complete graph. The pseudo-code of this algorithm is presented below.

Since every trapezoid will be added and removed exactly once from the modified binary indexed tree data structure, the total time complexity is $O(n \log n)$. This modified data structure with variable left ends is a novel approach to the best of our knowledge and makes this problem very interesting.

We conclude this section by summing the results in the following theorem.

Theorem 6 *The proposed algorithm calculates the vertex connectivity of a trapezoid graph with n vertices in time $O(n \log n)$ and space $O(n)$.*

4 Bipartiteness criteria and tree representation

In this section we establish local test for bipartiteness of trapezoid graphs.

Theorem 7 *The trapezoid graph G is bipartite if and only if it does not contain a triangle.*

Proof. The first part directly follows from the well-known result: the graph G is bipartite if and only if it does not contain odd cycles.

Let G be a triangle-free trapezoid graph and let $C = T[1]T[2] \dots T[k]$ be the smallest odd cycle contained in G with $k > 3$. It can be easily seen that there are no chords in C , i. e. there are no edges of the form $T[i]T[j]$ with $i > j + 1$ (otherwise we could find smaller odd cycle). Consider the intersection of the trapezoids $T[1]$ and $T[2]$. If their intersection is a trapezoid with height equal to the distance of two parallel lines (see the first part of Figure 4), then all trapezoids $T[3], T[4], \dots, T[k]$ must be on the right side of $T[1]$ – which is impossible, since $T[k]$ must have common points with $T[1]$. Otherwise, the trapezoids $T[1]$ and $T[2]$ have intersection as shown in the second part of Figure 4. Without loss of generality we can assume that the trapezoids $T[3]$ and $T[k]$ are independent and positioned as shown in the figure. In order to connect the trapezoids $T[3]$ and $T[k]$ by a path of trapezoids $T[4]T[5] \dots T[k-1]$ – some trapezoids of this path must intersect either $T[1]$ or $T[2]$, which is impossible.

Therefore, the graph G does not contain cycles of odd length and it follows that G is bipartite.

□

Note that from the above proof it follows that each cycle of length greater than four contains a chord (an edge joining two nodes that are not adjacent in the cycle).

A caterpillar graph is a tree such that if all pendent vertices and their incident edges are removed, the remainder of the graph forms a path. Let $C_{n,d}(a_1, a_2, \dots, a_{d-1})$ be a caterpillar

Algorithm 5: Calculating the vertex connectivity of a trapezoid graph.

Input: The trapezoids T , the arrays $leftmost$, $rightmost$ and $index_up$, and MBIT data structure.

Output: The vertex connectivity number.

```
1  $k = -1$ ;  
2  $left = 0$ ;  
3  $right = n$ ;  
4 for  $i = 1$  to  $n$  do  
5    $cut[i] = false$ ;  
6    $Update(c[i], 1)$ ;  
7    $Update(d[i], 0)$ ;  
8 end  
9 for  $x = 1$  to  $2n - 1$  do  
10   $i = index\_up[x]$ ;  
11  if  $a[i] = x$  then  
12     $cut[i] = true$ ;  
13     $right = right - 1$ ;  
14     $Update(c[i], 0)$ ;  
15     $Update(d[i], 0)$ ;  
16  end  
17  else  
18     $cut[i] = false$ ;  
19     $left = left + 1$ ;  
20     $Update(c[i], 0)$ ;  
21     $Update(d[i], -1)$ ;  
22  end  
23  if  $(x > 1)$  and  $(leftmost[x - 1] \neq -1)$  then  
24     $Update(d[leftmost[x - 1]], -1)$ ;  
25  end  
26  if  $(leftmost[x] \neq -1)$  then  
27     $Update(d[leftmost[x]], -n \cdot n - 1)$ ;  
28  end  
29  if  $(rightmost[x] \neq -1)$  and  $(leftmost[x] \neq -1)$  then  
30     $Nxy = Calculate(rightmost[x]) + n \cdot n + (n - right)$ ;  
31    if  $Nxy \leq n$  then  
32      if  $(k = -1)$  or  $(k > Nxy)$  then  
33         $k = Nxy$ ;  
34      end  
35    end  
36  end  
37 end  
38 if  $k > -1$  then  
39   return  $k$ ;  
40 end  
41 else // complete graph  
42   return  $n - 1$ ;  
43 end
```

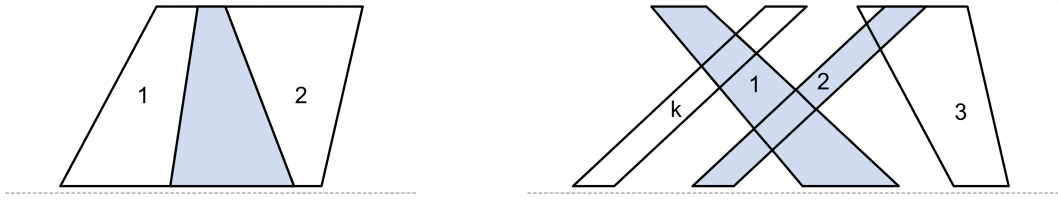


Figure 2: Two cases for Theorem 7.

with n vertices obtained from a path $P_{d+1} = v_0v_1 \dots v_{d-1}v_d$ by attaching $p_i \geq 0$ pendent vertices to the vertex v_i , $1 \leq i \leq d-1$, where $n = d+1 + \sum_{i=1}^{d-1} p_i$. It can be easily seen that each caterpillar $C_{n,d}(a_1, a_2, \dots, a_{d-1})$ has trapezoid representation as triangle-free interval graph [15].

Assume now that tree G is not a caterpillar and has trapezoid representation. Then it contains a vertex w with neighbors v_1, v_2, v_3 , such that each vertex v_i has another neighbor u_i different than w , $i = 1, 2, 3$. The trapezoids $T[v_1]$, $T[v_2]$ and $T[v_3]$ corresponding to the vertices v_1 , v_2 and v_3 are independent. Without loss of generality we can assume the order $T[v_1] \ll T[v_2] \ll T[v_3]$. Since all trapezoids $T[v_1]$, $T[v_2]$ and $T[v_3]$ intersect the trapezoid $T[w]$, it can be easily seen that all neighbors of $T[v_2]$ (trapezoid $T[u_2]$ in particular) also must intersect $T[w]$. This is a contradiction, and G does not have trapezoid representation. Therefore, we proved the following

Theorem 8 *A trapezoid graph G represents a tree if and only if it is a caterpillar.*

5 Concluding remarks

In this paper we presented an efficient algorithm for calculating the vertex connectivity number $\kappa(G)$ of a trapezoid graph. We leave as an open problem to design efficient algorithm for finding the edge connectivity number $\lambda(G)$ in trapezoid graphs.

The k -trapezoid graphs are an extension of trapezoid graphs to higher dimension orders. The k -dimensional box representation (V, l, u) of a graph $G = (V, E)$ consists of mappings $l : V \rightarrow R^k$ and $u : V \rightarrow R^k$ such that $l[i]$ is the lower and $u[i]$ the upper corner of a box $box[i]$ where two vertices of the graph are joined by an edge iff their corresponding boxes are incomparable [10]. If a graph has such a representation, it is a k -trapezoid graph. If we additionally have a weight $w : V \rightarrow R$ on the vertices of G then the k -trapezoid graph is weighted. For the case $k = 2$, we have simple trapezoid graphs.

Another generalization are circular trapezoid graphs – the intersection graphs of circular trapezoids between two parallel (concentric) circles [20]. It seems that the presented approach can be modified and adapted for calculating the vertex connectivity number of k -trapezoid graphs and circular trapezoid graphs.

Acknowledgement. This work was supported by Research Grants 174010 and 174033 of Serbian Ministry of Education and Science.

References

- [1] M. I. Abouelhoda, E. Ohlebusch, *Chaining algorithms for multiple genome comparison*, J. Discrete Algorithms 3 (2005) 321–341.
- [2] D. Bera, M. Pal, T. K. Pal, *An efficient algorithm to generate all maximal cliques on trapezoid graphs*, Int. J. Comput. Math. 79 (2002) 1057–1065.

- [3] D. Bera, M. Pal, T. K. Pal, *An Efficient Algorithm for Finding All Hinge Vertices on Trapezoid Graphs*, Theory Comput. Systems 36 (2003) 17–27.
- [4] F. Cheah, D. G. Corneil, *On the structure of trapezoid graphs*, Discrete Appl. Math. 66 (1996) 109–133.
- [5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press, 2001.
- [6] D. G. Corneil, P. A. Kamula, *Extensions of permutation and interval graphs*, In: Proceedings of 18th Southeastern Conference on Combinatorics, Graph Theory and Computing, 1987, 267–275.
- [7] C. Crespelle, P. Gambette, *Unrestricted and complete Breadth First Search of trapezoid graphs in $O(n)$ time*, Inform. Process. Lett. 110 (2010) 497–502.
- [8] I. Dagan, M. C. Golumbic, R. Y. Pinter, *Trapezoid graphs and their coloring*, Discrete Appl. Math. 21 (1988) 35–46.
- [9] S. Even, R. E. Tarjan, *Network flow and testing graph connectivity*, Inform. Process. Lett. 4 (1975) 507–518.
- [10] S. Felsner, R. Müller, L. Wernisch, *Trapezoid graphs and generalizations, geometry and algorithms*, Discrete Appl. Math. 74 (1997) 13–32.
- [11] P. K. Ghosh, M. Pal, *An efficient algorithm to find the maximum matching on trapezoid graphs*, J. Korean Soc. Ind. Appl. Math. IT Series 9 (2) (2005) 13–20.
- [12] P. K. Ghosh, M. Pal, *An efficient algorithm to solve connectivity problem on trapezoid graphs*, J. Appl. Math. & Computing 24 (2007) 141–154.
- [13] M. Hota, M. Pal, T. K. Pal, *Optimal sequential and parallel algorithms to compute all cut vertices on trapezoid graphs*, Comput. Optim. Appl. 27 (1) (2004) 95–113.
- [14] A. Ilić, A. Ilić, *On vertex covers and matching number of trapezoid graphs*, (2011), submitted.
- [15] E. Jürgen, *Extremal interval graphs*, J. Graph Theory 17 (1993) 117–127.
- [16] R. M. Karp, *Reducibility among combinatorial problems*, In: R. E. Miller, J. W. Thatcher (Eds.), Complexity of Computer Computation, Plenum Press, New York, 1972, 85–103.
- [17] Y. D. Liang, *Domination in trapezoid graphs*, Inform. Process. Lett. 52 (1994) 309–315.
- [18] Y. D. Liang, *Steiner set and connected domination in trapezoid graphs*, Inform. Process. Lett. 56 (1995) 101–108.
- [19] M. S. Lin, Y. J. Chen, *Counting the number of vertex covers in trapezoid graph*, Inform. Process. Lett. 109 (2009) 1187–1192.
- [20] Y. L. Lin, *Circular and circle trapezoid graphs*, J. Sci. Eng. Tech. 2 (2006) 11–17.
- [21] T. W. Kao, S. J. Horng, *Computing k -Vertex Connectivity on an Interval Graph*, International Conference on Parallel Processing ICPP’94, Vol. 3 (1994) 218–221.
- [22] T. H. Ma, J. P. Spinrad, *On the 2-chain subgraph cover and related problems*, J. Algorithms 17 (1994) 251–268.

- [23] G. B. Mertzios, D. G. Corneil, *Vertex splitting and the recognition of trapezoid graphs*, Technical Report AIB-2009-16, RWTH Aachen University, 2009.
- [24] S. Mondal, M. Pal, T. K. Pal, *An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs*, Int. J. Comput. Eng. Sci. (IJCES) 3 (2002) 103–116.
- [25] Y. T. Tsai, Y. L. Lin, F. R. Hsu, *Efficient algorithms for the minimum connected domination on trapezoid graphs*, Inform. Sci. 177 (2007) 2405–2417.